A framework for system-level co-verification using the BST infrastructure

Gustavo R. Alves, Telmo Amaral and José M. Martins Ferreira

Abstract

A good verification strategy should bring near the simulation and real functioning environments. In this paper we describe a system-level co-verification strategy that uses a common flow for functional simulation, timing simulation and functional debug. This last step requires using a BST infrastructure, now widely available on commercial devices, specially on FPGAs with medium/large pin-counts.

1. Introduction

System-level simulation enables to check the design correctness before any hardware is produced. If the system functionality is determined by a program stored in memory, then the term co-simulation is better applied. CPLDs and FPGAs are currently being used for system rapid prototyping. These devices, specially those with larger pin-counts, are being released with a BST infrastructure [1] for some years now. Development systems for these devices are now extremely powerful and versatile, enabling complex designs to be entered in multi-level forms (from schematic to hard/soft IP cores), simulated, synthesised, fitted, re-simulated (with delays) and finally programmed into one or more devices, either by using appropriated hardware platforms or just a simple cable connected to the PC serial/parallel port. Some devices, may be programmed/configured through the TAP, thus enabling quick and efficient in-system alterations.

In view of all these advantages, we proposed ourselves, in a recent system design to draw a system-level co-verification strategy that could, early in the specification phase, combine and explore the potentialities of the Altera Max+Plus II environment and the BST infrastructure that exists on all devices of the EPF10K family [2,3].

The system architecture is described in section 2 and the co-verification strategy is described in section 3. Section 4 presents the conclusions and the current status of our work.

2. The system architecture

Our system comprises two generic devices with an extended BST infrastructure, one emulating an 8-bit non-inverting unidirectional buffer (‘244) and the other emulating an 8-bit latch with tri-state outputs (‘373), a dual-processor controller, and two memories containing the program executed by each processor. One of the processors controls the extended BST infrastructure included in each generic device. As each one of these devices is to be implemented in an FPGA from the Altera EP10K Family, already containing a BST infrastructure, in the end the device will have two TAPs (one connected to the original BST infrastructure and the other one being part of our design). The second processor controls the system clock. It contains a group of 16 inputs and 16 outputs that can be used for any generic purposes.

Figure 1 illustrates the system architecture. ROM1 and ROM2 are used for simulation purposes, meaning that they do not correspond to FPGAs. The controller is implemented in an EPF10K30, and the ‘244 and ‘373 are implemented in two EPF10K10. Original TAP pins, power pins and other dedicated pins belonging to the device are usually not represented.

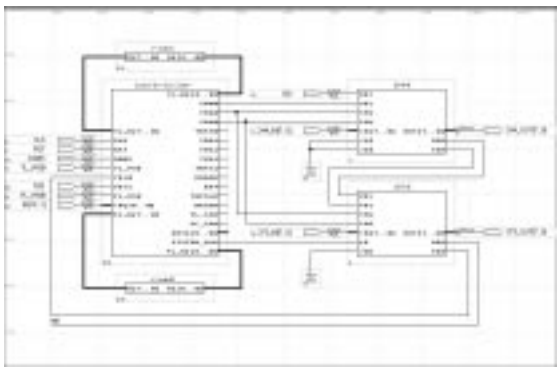

Fig. 1: The system-level architecture.

Our goal was to specify, develop and verify a system-level debug and test infrastructure based on a built-in controller and an extended BST infrastructure, that could be used for functional and timing debug. The built-in controller would be responsible for controlling the system BS chains and the system clock, thus guaranteeing the synchronism between the system functional and test logic. The extended BST infrastructure would provide support to Breakpoint (BP) and Real-time analysis (RT)

operations. For BP operations the BS register is configured to detect a condition corresponding to values present at the input pins or outputs from the component functional logic. For RT operations the BS register is configured to:

- Store a sequence of two contiguous vectors.
- Store a sequence of two contiguous vectors after a certain condition is found.
- Store a sequence of two contiguous vectors until a certain condition is found.

The system's functionality is described in another paper. In this document we will focus on the system co-verification strategy.

3. The co-verification strategy

In the system specification its was decided to set up a sound co-verification strategy that could address the following steps: functional simulation, timing simulation, and functional debug. Structural test was also part of the verification strategy, although it was address as an independent task, mainly done through the original BST infrastructure of the FPGAs, that unfortunately is not supported by the Max+Plus II model generation tool.

The system specification included the specification of each individual component, and some small debug and test programs to be executed by the dual-processor built-in controller, named PRODEP (PROcessors for DEbugging Purposes). The design of each component evolved in a mixed of top-down/bottom-up, block-based design, where some parts corresponded to previously developed blocks. For instance, one of the processors corresponds to an enhanced version of a board-level BIST processor [4]. The first verification stage corresponded to functionally simulating each component with a small number of hand-generated test vectors. This first pass enabled some confidence on the component's functionality, and also specific details to be thoroughly covered. Functional simulation of the controller included two stages, one similar to the previous one and another where the controller typically executed very small programs. To achieve this, a test environment comprising one controller and two memories was set up. This consisted of two devices, each containing one LPM, acting as a ROM, connected to the controller. Functional models of each device were first generated and a "system-level" linked functional model was then created using the capabilities offered by the Max+Plus II compiler.

The debug and test programs were written in assembly, and the object code and list files were then generated by a small freeware application that accepts table-defined instruction sets. A small in-house developed application then took the list file, and using a template file, produced a Memory Initialisation File (MIF) read by the simulation tool. The all process is illustrated in figure 2.

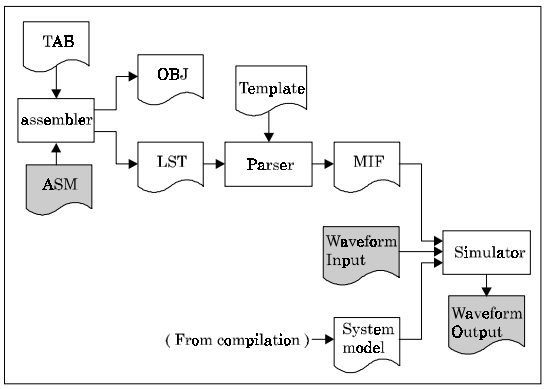

Fig. 2: Integration of an assembly file in the simulation process.

The second verification stage corresponded to a functional simulation of the all system. The functional models of each component were "linked" by the compiler and larger programs were written, assembled, and converted to MIFs, for performing the system-level co-simulation. After detecting, diagnosing and removing all design errors (at this stage) a golden simulation file with input stimulus and output vectors was produced, for later comparison during the timing simulation phase. A table file in ASCII format, containing the values for all system pins, was also generated. Fig. 5 illustrates these first design steps. The design flow then proceeded to the synthesis phase. Fig. 6 illustrates the following design steps.

After the synthesis and fitting process, a timing model of each component was generated for individual timing simulation by re-using the input stimulus. In the Max+Plus II waveform visualisation tool the values present on the component outputs were compared against those previously stored in the golden file, produced after the functional simulation stage. Although this was a manual process, if an error due to a long-path occurs, the component behaviour diverges significantly, and the detection is generally easy (due to the small length of individual simulation files). After removing all errors at this stage, the design proceeded to the system-level timing co-simulation.

A "linked" timing model of the system was first generated by the compiler, and the small test programs used during system-level functional simulation were now re-used. At the end of this verification stage, after removing all detected errors, a new table file was generated for automatic comparison with the table file generated during system-level functional simulation. This automatic process, illustrated in Fig. 3, required an

intermediate step where vectors not corresponding to moments relative to clock positive edges, were removed from the original table file. Comparing the two files corresponded to comparing the outputs generated during functional and timing simulation. This automatic process enabled larger programs to be written and verified, without recurring to tedious visual inspections on extensive waveform files.

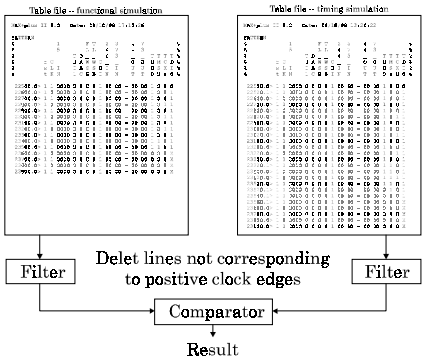

Fig. 3: Comparing two table files.

The next phase consisted of creating the board, programming the FPGAs, downloading the test programs (executed by the two processors) to the memories, and performing the system functional debug. This verification stage was carried out using the original BST infrastructure existing in the FPGAs and a PC-based application called TAPPER, able to control two BS chains. This application emulates the referred BIST processor, by executing the same instruction set, and controlling/reading the board TAP signal from the PC parallel port. Functional debug corresponds to verifying in-circuit the values obtained during functional simulation [5, 6, 7]. This is done in the following way:

1. TAPPER shifts the Sample/Preload instruction to all system devices
2. TAPPER places TAP controllers in *Select-DR*
3. The system primary inputs are externally fed with the right stimulus
4. One clock-pulse is applied to the system
5. TAPPER places TAP controllers in *Shift-DR*, through *Capture-DR* (where values appearing at component pins are captured in the BS register), and the first vector is shifted out and compared against the vector stored in one of the table files.
6. Repeat steps 2:5 until all vectors are compared.

The test program executed by TAPPER is generated by an in-house developed application that uses as input information: the BSDL files of the FPGAs, a configuration file, and the table file (for creating the expected values and masks for the comparison). The functional debug environment is illustrated in Fig. 4.

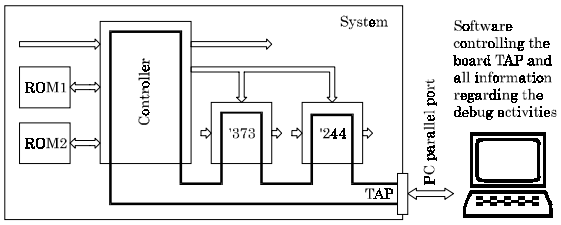

Fig. 4: Using TAPPER for system-level functional debug.

4. Conclusion and future work

In this paper we described a system-level co-verification strategy based on the potentialities of the Max+Plus II environment and the presence of a BST infrastructure on commercial FPGAs. Our strategy included four main verification phases: functional simulation, timing simulation, structural test, functional debug, and timing debug. Functional simulation provided initial information for the remaining phases. Vectors obtained during timing simulation were compared, at clock edges, with those generated during functional simulation. After simulating the all system, the design proceeded to the prototype phase, where functional debug took place on real hardware. The original BST infrastructure available on the FPGAs was used for sampling the values appearing at the component pins, on a vector-by-vector basis. Each vector was compared against the expected vector, extracted from the table files generated at the end of the functional simulation stage.

Although it may be confusing that our system contains two BST infrastructures, it should be reminded that we were developing a system-level debug and test infrastructure, and because the FPGAs used for prototyping purposes, already contained a BST infrastructure, in the end they co-existed, one implemented at the silicon foundry, and the other implemented by us. We used this approach for debugging our own debug and test infrastructure. In a near future, we plan to incorporate this debug and test infrastructure, based on the extended BST infrastructure and the built-in controller, in complex systems. This infrastructure provides an appropriate mechanism for system functional debug and timing debug. This last step (timing debug) was not performed on our prototyped system, because TAPPER does not run the test program at the same speed of the built-in controller, and because the original FPGAs' BST infrastructure does not match our extended BST infrastructure. Also, we could not use our controller for debugging our present system, because there would be an overlapping, causing potential conflicts on error location.

5. References

[1] IEEE Standard Test Access Port and Boundary-Scan Architecture, Oct. 1993, IEEE Std. 1149.1 (Includes IEEE Std. 1149.1a).

[2] The Max+Plus II Development System, http://www.altera.com/html/products/maxplus.html.

[3] The FLEX 10K Device Family, http://www.altera.com/html/products/f10k.html.

[4] J. M. Ferreira, J. S. Matos and F. S. Pinto, "Automatic Generation of a Single-Chip Solution for Board-Level BIST of Boundary Scan Boards," *EDAC Proc.*, March 1992, pp. 154-158.

[5] M. F. Lefébvre, "Functional Test and Diagnosis: A Proposed JTAG Sample Mode Scan Tester," in *ITC*, pp. 294-303, IEEE Computer Society Press, 1990.

[6] Richard M. Sedmak, "Boundary-Scan: Beyond Production Test," in *ITC*, pp. 415-420, IEEE Computer Society Press, 1994.

[7] Andy Halliday et al., "Prototype Testing simplified by Scannable Buffers and Latches," in *ITC*, pp. 174-181, IEEE Computer Society Press, 1989.

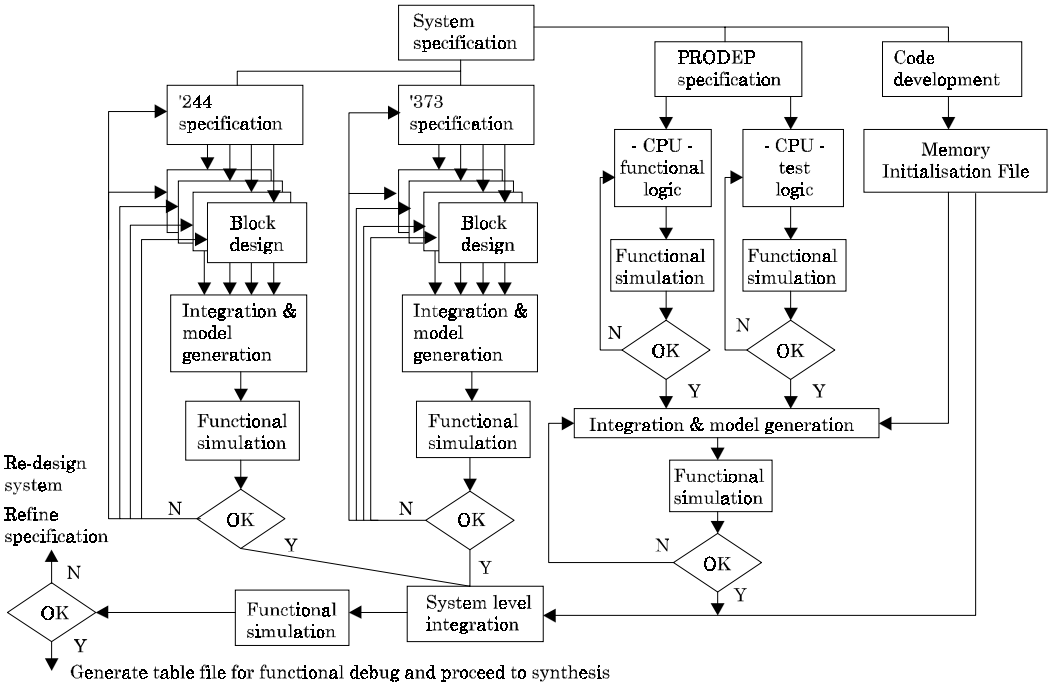

Fig. 5: Design steps leading to the system-level functional simulation.

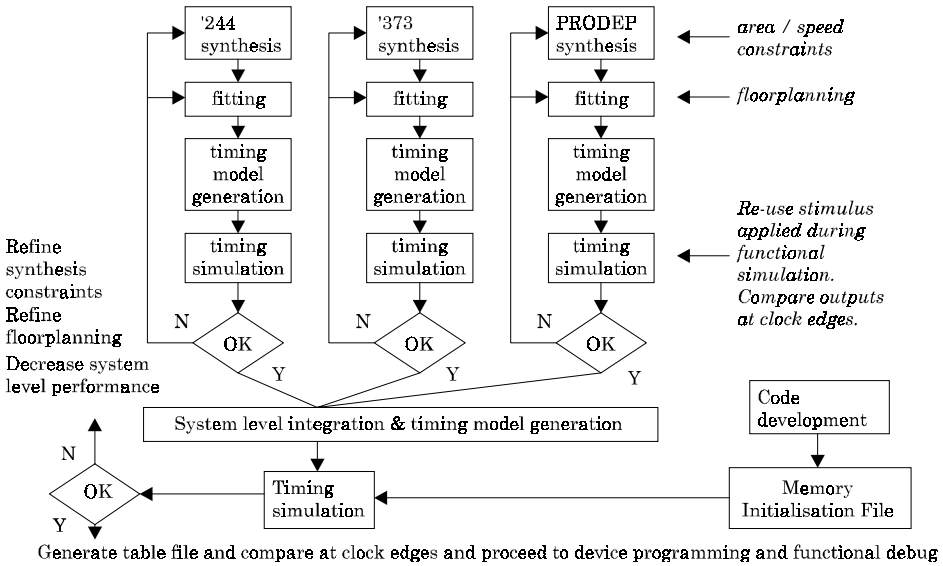

Fig. 6: Design steps leading to the system-level timing simulation.